PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : G06F 9/46, H04L 29/06

A1

(11) International Publication Number: **WO 97/30392**

(43) International Publication Date: 21 August 1997 (21.08.97)

(21) International Application Number: PCT/US97/02302

(22) International Filing Date: 14 February 1997 (14.02.97)

(30) Priority Data:
60/011,848 16 February 1996 (16.02.96) US

(71)(72) Applicants and Inventors: HOUSER, Shawn, W. [US/US]; 8650 West Monte Vista, Phoenix, AZ 85037 (US). DOTSON, Robert, E. [US/US]; 1303 E. Wescott, Phoenix, AZ 85024 (US).

(74) Agent: LANZA, John, D.; Testa, Hurwitz & Thibeault, L.L.P., High Street Tower, 125 High Street, Boston, MA 02110 (US).

(81) Designated States: AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CU, CZ, DE, DK, EE, ES, FI, GB, GE, HU, IL, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, TJ, TM, TR, TT, UA, UG, UZ, VN, YU, ARIPO patent (KE, LS, MW, SD, SZ, UG), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, ML, MR, NE, SN, TD, TG).

Published
With international search report.
Before the expiration of the time limit for amending the claims and to be republished in the event of the receipt of amendments.

(54) Title: METHOD AND APPARATUS FOR PROCESSING DATA

(57) Abstract

A data processing system is provided having one or more gateway processors, one or more controller processors which communicate with the gateway processors, and one or more processing engines which communicate with both the gateway processors and the controller processors. In one embodiment, one or more spooler processors are provided which communicate with the gateway processors, the controller processors and the processing engines. In another embodiment, one or more client processors are provided which communicate with the gateway processors. A method for processing data is also described in which a gateway processor is provided, the gateway processor receives a transaction from a client, distributes information to the controller processors in response to the transaction, and then responds to the client processor.

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AM	Armenia	GB	United Kingdom	MW	Malawi
AT	Austria	GE	Georgia	MX	Mexico
AU	Australia	GN	Guinea	NE	Niger
BB	Barbados	GR	Greece	NL	Netherlands
BE	Belgium	HU	Hungary	NO	Norway
BF	Burkina Faso	IE	Ireland	NZ	New Zealand
BG	Bulgaria	IT	Italy	PL	Poland
BJ	Benin	JP	Japan	PT	Portugal
BR	Brazil	KE	Kenya	RO	Romania
BY	Belarus	KG	Kyrgystan	RU	Russian Federation
CA	Canada	KP	Democratic People's Republic of Korea	SD	Sudan
CF	Central African Republic			SE	Sweden
CG	Congo	KR	Republic of Korea	SG	Singapore
CH	Switzerland	KZ	Kazakhstan	SI	Slovenia
CI	Côte d'Ivoire	LI	Liechtenstein	SK	Slovakia
CM	Cameroon	LK	Sri Lanka	SN	Senegal
CN	China	LR	Liberia	SZ	Swaziland
CS	Czechoslovakia	LT	Lithuania	TD	Chad
CZ	Czech Republic	LU	Luxembourg	TG	Togo
DE	Germany	LV	Latvia	TJ	Tajikistan
DK	Denmark	MC	Monaco	TT	Trinidad and Tobago
EE	Estonia	MD	Republic of Moldova	UA	Ukraine
ES	Spain	MG	Madagascar	UG	Uganda
FI	Finland	ML	Mali	US	United States of America
FR	France	MN	Mongolia	UZ	Uzbekistan
GA	Gabon	MR	Mauritania	VN	Viet Nam

METHOD AND APPARATUS FOR PROCESSING DATA

Field of the Invention

The present invention relates to systems and methods for processing data, and in particular, to systems and methods for processing data in parallel.

Background of the Invention

Since the inception of the electronic computer, much consideration has been given to processing data in parallel in order to speed up processing and reduce user wait time for results. Reduction of user wait time can be important for: querying a database in real-time, as in a license plate search; servicing multiple transactions at the same time, as in financial transactions; or producing complex images rapidly, as in medical imaging systems in which calculation and presentation of the desired image may have to be done as quickly as possible.

Early attempts at implementing parallel processing focused on analyzing a proposed algorithm for recursive structure and distributing the instruction execution of those recursive regions across a number of processors. While this method is fairly successful, it lacks flexibility and efficient scalability.

For example, in many cases the system architect was required to know in advance the number of processors necessary to effectively execute the algorithm. If the algorithm changed, the number of processors required for effective execution also changed. Because addition of processors in response to each software change was generally difficult and cumbersome, systems were typically provided with many processors in order to ensure the system could be used if the algorithm changed. This, of course, had the drawback of being prohibitively expensive.

Further, the addition of processors did not result in a linear increase in processing power because of overhead required to coordinate the processors. Thus, if the number of processors in a system were doubled, the speed of the system usually did not double because of communication and synchronization overhead.

Therefore, a need exists for parallel processing schemes that can be efficiently and inexpensively scaled. The present invention provides the aforementioned desirable characteristics while avoiding the undesirable characteristics of prior art devices.

Summary of the Invention

The invention relates to a system and method for processing data. The system has one or more gateway processors, and one or more processing engines which communicate with the gateway processors. In some embodiments of the system, it is desirable to have a controller processor, which communicate with both the gateway processors and the processing engines. In a typical embodiment of the system, one or more spooler processors are provided which communicate with the gateway processors, the controller processors and the processing engines. In another embodiment of the system that allows external control of the system, a client processor is provided which communicates with the gateway processors. One feature of the invention is that the gateway processors, the controller processors, the processing engines and the spooler processors, if provided, communicate with each other by exchanging messages. This allows the system of the present invention to be distributed across a network of machines or reside in the memory of a single machine. The system can, therefore, be scaled simply by adding additional processors or machines to the network.

A gateway processor is responsible for receiving a transaction from one of the client processors, distributing data to the controller processors in response to the client transaction, and responding to the client transaction. In other embodiments, the gateway processor may also distribute data to the processing engines or the spooler engines in response to the client transaction.

The above mentioned objects (the gateway processor, the controller processor, the spooler processor, and the processing engine) as described above are functional groupings of responsibilities and tasks that are distributed in the prescribed manner to maximize the flexibility and performance of the system. In the current implementations of the system these particular functional groupings are used to build a commercially applicable data processing system from the system components. It is an important feature of the system that groupings such as these can be created and distributed in any manner to most effectively utilize the underlying physical

componentry of the system. This is made possible by the system's ability to decompose larger tasks into smaller idempotent distributable and parallelizable sub tasks.

Brief Description of the Drawings

This invention is pointed out with particularity in the appended claims. The above and further advantages of this invention may be better understood by referring to the following description taken in conjunction with the accompanying drawings, in which:

Fig. 1A is a diagrammatic view of an embodiment of a data processing system of the present invention servicing a single client transaction;

Fig. 1B is a diagrammatic view of an embodiment of a data processing system of the present invention servicing multiple client transactions;

Fig. 2 is a diagrammatic view of an embodiment of a data processing system of the present invention servicing a transaction in which there is a conflict for a particular processing engine; and

Fig. 3 is a diagrammatic view of an embodiment of a processor of the present invention.

Detailed Description of the Invention

In overview, the data processing system of the present invention includes at least one gateway processor 12, at least one controller processor 14, and at least one processing engine 16. If output is desired from the system, then at least one spooler processor 18 may be provided. Additionally, a client processor 10 may be provided to facilitate communication between user applications and gateway processors 12. Generally, any number of processors may be used in any configuration. For example, FIG. 1A shows an embodiment of a data processing system including a client processor 10, a gateway processor 12, a controller processor 14, a processing engine 16, and a spooler processor 18. FIG. 1B shows an embodiment of a data processing system including two client processors, generally 10, four gateway processors, generally 12, two controller processors, generally 14, four processing engines, generally 16, and three spooler processors, generally 18. The various processors of the systems shown in FIGS. 1A and 1B may be implemented in hardware, may be implemented as software processes resident in the memory of a single computer, or may be implemented as software processes which are distributed among a

network of computers. For ease of reference throughout the specification, the generic term "processor" will be used when referring to the various processors of the system. However, it should be understood that use of the term "processor" encompasses hardware processors as well as software processes.

If the processors are distributed over a network of computers, the processors may communicate by way of a communication line which may be: a proprietary data bus; a standard data bus, such as SCSI, NUBUS, ISA, PCI, HIPPI, or EISA; or a network protocol such as TCP/IP, Ethernet, or FDDI. In a preferred embodiment, each processor is implemented in software resident on a personal computer which is networked to other personal computers which host the other processors included in the system.

The various processors communicate with each other through messages. The messages can have any format which allows for unambiguous communication between processors. In a currently preferred embodiment, messages sent between processors must begin with a four-byte length field. This field indicates the length of the message being sent. The length field may be followed by a four-byte transaction code, which is a unique identification code specifying the type of work required of the receiving processor in order to respond to the message. The transaction code is followed by the message. The message can include any information associated with the message or it can be data. Typically, there is enough information contained in the message for the system's communications layer to transparently support replication and system pairing capacities, without the explicit knowledge of higher level applications. The message passing is optimized to perform only those transmissions that are needed. For example, many transmissions are simply broadcast without the need for establishing a session connection or receipt verification. The system as a whole uses the message layer as a central nervous system, tying together and synchronizing what are often thousands of separate subtasks that form a single request from a client.

The Client Processor

A client processor 10, if provided, is responsible for accepting requests from a user application, formatting those user application requests into messages suitable for transmission over the communication line that interconnects the processors of the system, and transmitting the request message to the gateway processor 12. The client processor 10 may be an integral part of

a user application. Alternately, the client processor may be provided as a separate entity that interfaces with multiple user applications. Currently preferred is providing the client processor 10 as a dynamically linked library which can be used concurrently by various user applications. This dynamically linked library format is currently used to provide non-proprietary interfaces to the system, by supporting facilities such as ODBC and X/Open.

The client processor 10 may be able to format messages suitable for transmission using various communication line protocols in order to act as a rudimentary form of bridge between processors connected by different communication lines. However, currently preferred is a client processor 10 that formats user application requests suitable for transmission using only one communication line protocol.

The client processor often acts as a translation layer, converting host specific data to and from the system's native storage formats. Additionally, the client may perform various sorts of validations of application inputs, and even extends new features to support a wider range of clients. The client processor, like the other processing engines, has full access to the functional operations primitives of the system. The client processor in some implementations, therefore, is able to perform a substantial duty in the form of off-loading processing overhead from the objects that comprise the central system, enhancing the system's scalability.

Once the user application request message is properly formatted for transmission, the client processor 10 selects a gateway processor 12 to which it transmits its request. Selection of the gateway processor 12 can be done in any number of ways. It is better, for the sake of efficiency, that user application requests are fairly evenly distributed among gateway processors 12. One way of achieving such "load leveling" is that a client processor 10 may, before transmitting a request, determine the "loading", i.e., the activity level, of any particular gateway processor 12 before transmitting its request. Alternately, each client processor 10 may maintain an ordered list of gateway processors 12 which it

transmits requests to in round-robin order. Currently preferred, however, is for each client processor 10 to maintain a list specifying a primary gateway processor 12 and a list of alternate gateway processors 12. The client processor 10 of the preferred embodiment transmits requests to its primary gateway processor 12, and only transmits requests to the alternate gateway processors 12 when its primary gateway processor 12 is non-responsive.

A user application request message may include a request for output; for example, a user application request may be a database query that also includes a request for report output. The request for output will contain additional information that may be extremely detailed; for example, the report requested by the user may specify the width of the report columns, the font to be used, etc. At the very least, however, the client processor's output request includes the identity of the spooler processor 18 which should service the output request. The client processor 10 may select the identity of the desired spooler processor 18 in any number of ways. In general, however, a spooler processor 18 is selected based on the type of output device it controls. For example, the client processor 10 may inquire of all the spooler processors 18, at the time of the user application request, what type of output device the respective spooler processor 18 is controlling. The client processor 10 would then use the returned information to select the identity of the desired spooler processor 18. Currently preferred, however, is for the client processor 10 to make this inquiry at system start-up time and store the returned information in a list, which it accesses when it formats the user application request. As an alternative to the above routine, the client may also request the return of its query in a raw data format, which may be gathered and returned by a spooler or gateway processor as may be appropriate.

The Gateway Processor

A gateway processor 12 is responsible for receiving user application requests and distributing work to other processors in the system in order to service those user application requests. A gateway processor 12 receives user application request messages, whether sent by a client processor 10 or sent directly from a user application. Although it is contemplated that a gateway processor 12 may receive incoming user application request messages from random client processors 10, it is currently preferred that each gateway processor 12 is assigned to one or more client processors as a primary gateway processor 12 and to one or more client processors as an alternate gateway processor 12. Thus, in the currently preferred embodiment, a single gateway processor 12 may receive user application request messages from more than one client processor 10, but only if multiple client processors 10 have included gateway processor 12 in their list of primary and alternate gateway processors to which requests may be sent.

When a user application request received, the gateway processor 12 assigns a unique session identifier to the user application making the request. Thus, user applications capable of

making multiple simultaneous requests will be assigned multiple, unique session identifiers associated with each individual request. Once a session identifier is assigned, the gateway processor 12 transmits that session identifier to the user application making the request. Any mechanism for ensuring the session identifier is unique may be used. For example, a list containing all active session identifiers may be stored in memory that must be checked by all gateway processors 12 before assigning a session identifier. Alternately, each gateway processor 12 could be required to maintain a table of assigned session identifiers, the table being shared with all other gateway processors 12. Currently preferred, however, is assigning a set of session identifiers to each gateway processor 12 which only that particular gateway processor 12 can assign.

Session identifiers for the currently preferred embodiment may be formed in any fashion, so long as each gateway processor 12 does not form or assign session identifiers outside of the range assigned to it. In the currently preferred embodiment, a gateway processor 12 multiplies its own identification number, which is stored in its memory, by a predetermined constant, which is generally equal to the maximum number of session identifiers each gateway processor 12 is allowed to assign. The gateway processor 12 then increments an integer variable. The integer variable is added to the product described above unless incrementing it caused it to equal the predetermined constant, in which case is set to zero before it is added to the product.

Each gateway processor 12 maintains a list of active session identifiers which it has already assigned to user application requests. Once the gateway processor 12 has formed a session identifier, it compares the session identifier against its list of active session identifiers. If the formed session identifier does not appear in the list, it is assigned. However, if the session identifier so formed does appear in the list, a new session identifier must be formed. In order to form a new session identifier, the gateway processor 12 repeats the calculation described above and checks the new session identifier against its list of active session identifiers. This process repeats until a session identifier is formed which does not appear in the list of active session identifiers maintained by the gateway processor 12.

For example, in order to form a unique session identifier, "Gateway Processor 0" would multiply its own identification number, in this case 0, by a predetermined constant. An integer, e.g. 2, is then added to the product. This results in a session identifier of two for the present

example. The gateway processor 12, in this case "Gateway 0," then checks its list of active session identifiers to determine if "2" is present, and therefore active and previously assigned. If "2" appears in the list, it signifies that a user application request that has been assigned that session identifier is still active. In this case, the gateway processor 12, "Gateway 0," would increment the integer to three. If the predetermined constant was equal to three, the integer would be reset to zero. That is, the integer resets to zero whenever incrementing would cause it to exceed the predetermined constant. Assuming the predetermined constant is greater than three, the gateway processor 12 adds three to the product of its identification number and the predetermined constant, and checks its table of active session identifiers for the resultant session identifier, in this case "3." This procedure is repeated until a session identifier is formed that does not appear in the gateway processor's 12 list of active session identifiers. That session identifier is assigned to the user application request and is entered in the list of active session identifiers maintained by the gateway processor 12.

Once a session identifier has been assigned to the user application request, the gateway processor 12 assigns a unique task identification number to the user application request. Again, any method for ensuring the uniqueness of the task identification numbers may be used, although preferred is forming the task identification numbers by concatenating the gateway identification number, the current date, and the current time. It is evident, therefore, that one session identifier may be associated with multiple task identification numbers, i.e. one user application request identified by a unique session identifier may transmit multiple request messages, each of which require a unique task identification number.

Once a unique task identification number has been assigned to the user application request message, the gateway processor 12 enters the task identification number in a list of active task identification numbers and transmits the task identification number, the associated session identifier, and any other information associated with the user application request to a controller processor 14. It is contemplated that a gateway processor 12 may be assigned to one controller processor 14 to which it transmits data or that the gateway processor 12 could select one of the set of controller processors 14 to transmit to based on the controller processor's level of activity. Preferred, however, is for the gateway processor 12 to transmit information to a random controller processor 14. Any method of randomly selecting controller processors 14 may be used. In the currently preferred embodiment, each gateway processor 12 maintains a pointer which

points to the controller processor 14 that should receive the next message. At system start-up time, each gateway processor 12 sets its pointer to point to a controller processor 14 selected at random. Thereafter, each time a message is sent to a controller processor 14 by the gateway processor 12, the pointer is incremented to indicate that another controller processor 14 should receive the next message.

If the user application request message includes requested-output-format information, the gateway processor 12 transmits the task identification number and requested-output-format information to the spooler processor 18 indicated by the client processor 10 in the user application request message. If the selected spooler processor 18 is non-responsive, the gateway processor 12 transmits a message to the user application indicating that the report cannot be started. The user application then makes a decision whether to abort the report request or retry the report.

Once the spooler processor 18 has received the task identification number and requested-output-format information, the gateway processor 12 sends the user application request message to a processing engine 16 for processing. Processing engine 16 can be selected in any fashion as long as the request is taken by a processing engine 16 which has access to the data specified in the user application request message. The gateway processor 12 may broadcast the request to all processing engines 16.

However, it is currently preferred that each gateway processor 12 calculates this information for itself in the following manner. Each gateway processor 12 accesses a table definition file. The table definition file is created at start-up time and resides in the memory or on the disk of the gateway processor 12. The table definition file contains assorted information about each table on the system, including a mapping based on a data attribute. For example, in a database system, the table would provide a processing engine 16 mapping based on a primary key, i.e. a hash table. If the user application request is based on the attribute used for the mapping, the gateway processor 12 can use the mapping to determine which processing engines 16 need to be accessed. For example, if the user application is requesting a database sort based on the primary key contained in the table, the gateway processor 12 uses the hash table to determine which processing engines 16 have access to the requested data.

If, however, the user application requests data that is based on some other attribute of the data, the gateway processor 12 must calculate a new mapping using the table already provided. For example, if a user application requests data based on some other field which is not the primary key, the gateway processor 12 must calculate a new hash table using any method known in the art for generating a hash table with a level of indirection. The new hash table calculated by the gateway processor 12 indicates which processing engines 16 have access to the requested data; thus, the gateway processor 12 directs the user application request to the proper processing engine 16 based on the data affected by the user application request.

If a processing engine 16 indicated by the above determination is non-responsive, the gateway processor 12 determines if the user application request is directed to data that is redundantly stored. This determination can be done in many ways. Currently preferred is providing a one-byte flag in the table definition file. The value of the byte flag indicates whether or not the data is stored redundantly, as well as the level of redundancy.

If the data is stored redundantly, the gateway processor 12 transmits the user application request to the redundant processing engine 16. If the data is not stored redundantly, or if the processing engines 16 indicated as redundant are non-responsive, or if the gateway processor 12 returns a message to the user application indicating that its request has failed. The user application then decides whether to retry its request or abandon it. If the user application retries the request, the gateway processor 12 attempts to contact the indicated processing engines 16 again. If the user application aborts the request, the gateway processor 12 transmits a message that the request has aborted to any controller processor 14. This may be done because the controllers share data with each other. The gateway processor 12 then removes the task identification number from its list of active task identification numbers.

After transmitting the user application request message to the processing engines 16, the gateway processor 12 waits to receive a message indicating that the processing engine 16 has completed the user application request. In cases requiring that multiple processing engines 16 simultaneously perform an operation, the gateway processor 12 waits until it receives completion messages from all processing engines 16 required to perform an operation in response to the user application request. While the gateway processor 12 is waiting for completion messages, it continues to receive and distribute user application request messages from client processors 10.

Once all completion messages for a given task are received by the gateway processor 12, it transmits a message to the controller processor 14 currently indicated by its pointer. The message informs the controller processor 14 that the task associated with the user application request's task identification number has been completed. The gateway processor 12 then removes the task identification number from its list of active task identification numbers.

The gateway processor, because of its role as an external interface, often forms the root of what is defined in the invention as a "chained process". Often, a request made to the gateway is complex, and requires input from multiple other processing engines. In such a case, a status block associated with the task identification number is maintained. As each task is distributed, an entry is noted in this status block. As each task responds, often with a message of successful completion, failed completion, or a preparation to for final "commit" of a task, a notation is made within this status table. This allows for the decomposition of a task into a very large number of smaller atomic tasks that can be performed in parallel, yielding a much lower overall task completion time. Note that there is no need to monitor each task; instead, the task that notes each action on the task has the ability to take further action on the task. Further, the task is passively monitored for timeout. The chained process is not limited in any way to the gateway processor, it simply occurs there more often. As another example, a relational table with a foreign key on a processor engine node may need to retrieve a tuple form another processor engine node without pausing in its own task; it will therefore spawn a chained task to execute this operation in parallel. Due to the distributed nature of the system, the chained process forms a central methodology for the completion of complex tasks.

The Controller Processor

The controller processor 14 acts as a central synchronization and redundancy point for the data processing system of the present invention. As noted above, when a gateway processor 12 receives a user application request message, it assigns a unique task identification number to that request. The gateway processor 12 then transmits the task identification number and the associated session identifier to the controller processor 14 currently indicated by its pointer. The controller processor 14 stores this information in a list containing all active task identification numbers and then communicates it to all other controller processors 14 in the system. This may be done by a broadcast transmission from one controller processor 14 to all other controller

processors 14 present in the system or the receiving controller processor 14 may transmit the information to all other controller processors 14 individually. Preferred, however, is for one controller processor 14 to pass the information along to one other controller processor 14, which in turn does the same. Thus, at any one time, all controller processors 14 have a complete list of all active task identification numbers, the session identifiers with which they are associated, and any other associated information such as requested output information. Each controller processor 14, therefore, has a complete list of all active tasks that must be completed by the system.

When a user application request is completed or aborted, a controller processor 14 receives a message from a gateway processor 12 informing it that the task associated with a task identification number has either completed or aborted. The controller processor 14 relays this information to all other controller processors 14 as noted above. If requested output information is included in the user application request, the controller processor 14 forwards the request completed/request aborted message to the proper spooler processor 18, which is determined by the controller processor 14 in the same manner as the gateway processor 12. Preferably, the controller processor 14 determines the proper spooler processor 18 from the user application request message sent by the client processor 10. Additionally, each controller processor 14 maintains a set of system logs.

At system start-up time, one controller processor 14 is selected as "Controller 0." This may be the controller processor 14 that is actually "Controller 0," or, if the controller processor 14 designated as "Controller 0" fails to become active at system start-up time, the controller processor 14 having the nest highest rank, i.e. "Controller 1," assumes the duties of "Controller 0." Each processor is required to "check-in" with "Controller 0" when the processor first starts up. This allows "Controller 0" to create a comprehensive map of the system, i.e. the number of gateway processors 12, the number of controller processors 14, the number of processing engines 16 and the number of spooler devices 18 present.

In addition, "Controller 0" acts as a sanity check for the other controller processors 14 in the system. When "Controller 0" first starts up, it checks its log to determine if the system is starting up from a normal shutdown. If a normal shutdown is indicated, "Controller 0" contacts every other controller processor 14 in the system to verify that the last time-stamp in the log

maintained by "Controller 0" is the most recent time-stamp of any log. If this is true, the system starts up normally.

If "Controller 0" is unable to verify that its logs have the most recent time-stamp, or if the log maintained by "Controller 0" does not indicate a normal system shutdown, "Controller 0" must contact each controller processor 14 and request task identification numbers that were active for time period between the last successfully completed task and the last time-stamp in the system. "Controller 0" must check the validity of each of the task identification numbers so gathered. If all of the task identification numbers are valid, the system can start up. Otherwise, "Controller 0" must identify every invalid transaction and execute a series of transactions to restore the system to the state it was in before the invalid transaction began processing by executing transactions which "undo" the effects of the invalid transactions.

The Processing Engine

The processing engine 16 is largely responsible for storing the data present in the system and actually doing the calculations requested by the user application request message. Each processing engine 16 maintains at least a portion of the system's data, and each processing engine 16 maintains the following logs. A processing engine 16 may maintain the system's data on any memory device to which it has access. Thus, a processing engine may maintain the system's data on a single hard disk, a redundant disk array, a logical disk (i.e. a stripe of a larger, system-wide disk array), a floppy disk, a CD-ROM, an optical disk, random access memory, a tape drive, etc. In short, the processing engine may maintain system data on any type of memory device. A currently preferred method is for the processing engine to store data on "logical drives", which consist of several physical storage devices such as those mentioned above, with a suitably large cached subset of information maintained in random access memory. The processor stores the data in this way transparently to other requesting processes, and this methodology allows it to realize benefits in parallel throughput from the devices without requiring any specialized hardware to do so. Thus, multiple processing engines 16 may be provided that each contain the entirety of the data processing system's data. This configuration would allow a high level of fault tolerance which is desirable when data must always be available. Alternately, data could be apportioned among the various processing engines 16 in order to speed up actions on large amount of data. Another alternative would be to apportion the data among several processing engines 16 and then

replicate critical portions of the data in order to provide both speed as well as fault tolerance with respect to the redundant portion of data.

The Spooler Processor

The spooler processor 18 is responsible for producing output for the data processing system. A spooler processor 18 controls an output device. The output device controlled by the spooler processor 18 can be a printer, disk drive, display, or other device for outputting data in physical form. Additionally, a client processor 10 can request that the spooler processor 18 return data directly to it.

A spooler processor 18 is notified when a gateway processor 12 receives a user application request message that includes requested output information. As noted above, the gateway processor 12 sends the task identification number assigned to the user application request to the spooler processor 18 along with any requested output information. The spooler processor 18 stores the task identification number and the requested output information in a list. The spooler processor 18 waits until it receives a request completed/request aborted message from a controller processor 14 indicating that the user application request has finished. If the message received by the spooler processor 18 is a request aborted message, the spooler processor 18 removes the task identification from its list of reports to generate. If the message is, instead, that the user application request has completed, the spooler processor 18 requests the processed data from all of the processing engines 16. This may be done as a broadcast to all processing engines 16. Currently preferred, however, is requesting the data from the processing engines in round-robin fashion. As the spooler processor 18 receives data from the various processing engines 16, the spooler processor 18 formats the data according to the requested output information, and sends the formatted data to its associated output device. If the client processor 10 requested that the data be returned directly to the user application, the spooler processor 18 sends a message or messages to the client processor 10 containing the requested data.

For systems in which more than sixteen processing engines 16 are provided, it is currently preferred that one leader is selected for each group of sixteen processing engines. Each leader requests data from the other 15 processing engines in its group and formats the data for its group. The spooler processor 18 requests data from the leaders in round-robin order and does the final processing required to provide report output.

This scheme can also be used in applications in which the additional processing power of the processing engines 16 may be effectively used. For example, if the output requested is a sum total of various data entries distributed over various processing engines 16, it is preferred that each processing engine 16 functioning as a leader sum the data entries of every processing engine 16 in its group of sixteen and pass only the sum to the spooler processor 18. The spooler processor 18 then needs only to sum the responses from the "leader" processing engines 16 to provide the output requested by the client processor 10.

System Operation

Referring to Table 1 and FIGS. 1A and 1B, the data processing system of the present invention is shown servicing client transactions. A client processor 10 sends a request (step 102) that is received by a gateway processor 12 (step 102). If multiple client processors 10 are provided, they may make simultaneous user application requests, as shown in FIG. 1B. Requests sent by the client processors 10 may be database query requests that have associated report output requests or they may be simply requests to perform an operation on data. A client processor 10 can be a user application program that includes the proper mechanism for communicating with a gateway processor 12. Alternately, a client processor 10 can be a separate program which receives requests from an application program and forwards those requests to the gateway processor 12.

Each gateway processor 12 receiving a user application request assigns a unique session identifier to the user application request (step 104) and a unique task identification number to the specific request made by the client processor 10 (step 106). The gateway processor 12 sends the assigned session identifier to the user application (step 108), which then uses the unique session identifier to make subsequent, associated requests or check the status of a pending request. The gateway processor sends the task identification number to a random controller processor 14 (step 110), and enters the session identifier in its list of active session identifiers and the task identification number in its list of active task identification numbers. If more than one controller processor 14 is provided, as in FIG. 1B, the controller processors 14 now share task identification information (step 112).

The client transaction shown in FIGS. 1A and 1B includes a report output request. So, after the task identification number is sent to the controller processors 14, the gateway processor

12 sends the task identification number with the report request to a spooler processor 18 (step 112). The spooler processor 18 to which the report information is sent is selected by the client processor 10 on the basis of the type of output device the spooler processor 18 controls.

The gateway processor 12 determines the path necessary to access the data requested by the user application request and uses that information when accessing the table definition file, which was created at system start-up, that indicates which processing engine 16 has responsibility for the affected data. If the user application request requires exclusive access to the affected data, the gateway processor 12 locks the path to the affected data. The gateway processor 12 then sends the query request to the processing engines 16 (step 114). A user application request may require accessing multiple processing engines 16 or it may require accessing one processing engine 16 only. The processing engines 16 process the request (step 116) and notify the gateway processor 12 (step 118) when processing is complete. The gateway processor 12 forwards that information to the controller processors 14 (step 120) and unlocks the path. The controller processor 14 then notifies the spooler processor 18 that processing is complete (step 122).

During this processing several important decisions will be made by software. First, the gateway will analyze all methods known to it for solving the given client request. For each method, it will analyze the estimated time for completion. Each step, even to the addition of two simple integers, will be examined to determine the best course of action. When all known methods have been analyzed, the gateway processor will select the best and proceed with the operation.

Each piece of the processing that is performed by the system is made to be atomic in nature, and in effect makes the entire task atomic; so that the entire task or any subtask can be made either completely successful or completely nonexistent, with any side effects from processing removed.

Each process is broken down into "primitives" by the system, which are then compiled in a native machine language in the host machine's memory for execution. Once complete, this is termed a procedure, and this this procedure is built for reentrant and repeatable execution by multiple system processes and threads, allowing for more efficient use in high volume systems for repeat operations.

It is a function of the platform to allow each process as described to execute as if it was the only process currently executing on the system. In order to do this, is necessary to avoid the effects of the wormhole theorem by providing a true linear timeline for each process to execute by. The distributed nature of the system makes this especially difficult to accomplish this. Therefore, a unique mechanism is used by the system to resolve these dependencies. As each process is started, it receives a timestamp of the start of its execution from the gateway that started it. Note that although the gateway times may not be exactly in synch with each other, it is not important that the process timeline be linear to any specific timeline, just that it be exactly linear in and of itself while arbitrating with other timelines (processes). This timestamp, then, becomes its linear reference. As the system arbitrates access between the processes for each piece of information, it utilizes this timestamp to evaluate its course of action. First, the type of action is considered. For example, a read only action will not create the same dependencies for other processes that a write operation will. To allow the greatest amount of parallelism therefore, each operation is "queued" against the data in the event of parallelism. If a read operation follows a write operation on the linear timeline, it is allowed to see the data the incomplete write operation will write, and can move on to complete its own task. A write operation following a read operation, appropriately, will not have its modification made until the formal completion of the read operation. Complications arise when, for whatever reason, a process that is ahead in the linear timeline of other processes is not able to complete, therefore invalidating assumptions made by the other processes queued on the data item. To resolve this, the system is able to reexecute each operation in the queue for the arbitrated item. Importantly, it is nearly always able to do this without any other arbitration or communications. It does this by storing a predicate, or truth evaluation, and transform, or change instructions for each operation that is queued. It is therefore able to recreate each process's access to the data without any awareness on behalf of the process, and with very little execution overhead. The actual granularity of the arbitration may be a file, page, tuple, attribute, range, or any other object designation. This combination of locking concepts is unique, and gives the system an extraordinary capacity to perform ACID operations with a very low transaction cross execution time.

Continuing with the task being described, the spooler processor 18 requests output data from all of the processing engines 16 (step 124), formats the requested report based on the report

information forwarded to it by the gateway processor 12, and outputs the report to its associated output device (step 130).

Referring now to Table 2 and FIG. 2, the data processing system of the present invention is shown servicing two simultaneous user application requests that require access to the same data. Operation for both requests is similar to that described above in connection with FIGS. 1A and 1B until the second gateway processor 12 recognizes that its access path has been locked by the first gateway processor. When that occurs, the second gateway processor 12 is prohibited from making its request until the processing engine 16 is done processing and the gateway processor 12 unlocks the path. The second gateway processor 12 is then free to make its request.

Processor Implementation

The processors of the present invention may be implemented as hardware devices, mixed hardware and software devices or software devices. For example, a gateway processor 12 may be implemented as: a custom Application Specific Integrated Circuit providing the proper functionality; a microprocessor programmed to function properly; or a software process running on a general purpose machine. Referring now to FIG. 1, a processor of the present invention is shown implemented as a software device. FIG. 1 shows a parent process 40, which has spawned threaded child processes 42, 44, and 46. Threaded child processes 42 are the processes that actually carry out the work associated with a processor, whether the processor is a client processor 10, a gateway processor 12, a controller processor 14, a processing engine 16, or a spooler processor 18. Threaded process 44 is a statistics monitor. Threaded process 46 is a "watchdog" monitor. The child processes, shown generally as 42', do the work of the parent process 40. For example, a gateway processor 12 may have multiple client processors 10 making requests. For each client processor request, the gateway processor assigns a thread of execution, represented by child 42', which handles the set of steps associated with servicing that request.

Child process 44' monitors statistics associated with the operation of the system. For example, child process 44' may track the number of requests made to a processing engine 16, the percent utilization of various processors, the percent utilization of a disk drive associated with its parent process, the number of messages being transferred over the network or bus, the number of transactions the system is currently executing, or the error rate of messages transmitted between processors.

Child process 46' is a "watchdog" monitor. Its function is to monitor outstanding requests that exceed the processor's time-out period. When this occurs, the watchdog monitor terminates the child process associated with the task and restarts the task. For example, if a gateway processor 12 has a child process 42' that is making a request of a processing engine 16 and the processing engine 16 is shut off in the middle of the request, child process 12' will exceed the time-out period of the gateway processor 42. When this happens, monitor 46' terminates the child process 42' responsible for executing the request. Thread process 42' recognizes that its child process 42' has been terminated, and notifies the monitor 46' that child process 42' has terminated. The monitor 46' then restores any system-wide tables or variables affected by the child process 42' and transmits a message to the system that the task being executed by the child process 42' has finished. Monitor 46' then spawns a new child process 42' to attempt the request again.

Each child may in fact perform the work of any other child if so instructed, and this is an important feature of the system. The children themselves are "stateless robots", a term used in the invention to describe threads of execution that have no inherent memory or state of an operation. The children themselves, therefore, can be assigned at will to any given operation. This design allows a child to execute a process up until the point at which there will be a delay, such as a tuple retrieval from another node. At that point, it simply starts another task, or goes to work monitoring the incoming request queues. When a child receives a piece of information for a task, it also has the ability to immediately start executing that task should the received information allow for renewed processing. Any process memory that may be needed is stored with the task identification number, but often a task seems to simply take on a life of its own, moving from one node to the next in operation in a chained process as described earlier. This design is important because it facilitates the decomposition of single user requests into a number of parallelizable subtasks, and provides other benefits such as near zero wasted CPU cycles, since it is rare for a process to be waiting on any event. In the current embodiment of the system, the children are given specific instructions and grouped into functional objects manifested in the gateway processor, the controller processor, the spooler processor, and the processing engine.

Additionally, certain fundamentals must be in place in order to facilitate the levels of parallelism achieved in the system. For example, there are no runtime data objects that are not scalable for arbitration. This requires certain unique data structures. For example, a fundamental

system construct is a set of free memory blocks. In a normal system, all processes would arbitrate to get the next free block. In a high traffic situation, this can bring a multiple CPU system to a crawl. The system resolves this by scaling the pool into a number of segments set to distribute arbitration. The same free pool is used, but the number of arbitration points is scaled so that the number of requests at each point no longer interferes with system throughput. Such fundamental structures are needed to provide true scalability throughout the system.

Although only preferred embodiments are specifically illustrated and described herein, it will be appreciated that many other modifications and variations of the present invention are possible in light of the above teachings and within the preview of the appended claims without departing from the spirit and intended scope of the invention. Other objects, features and advantages of the invention shall become apparent when the following drawings, description and claims are considered.

Table 1

Step Number	*Client Processor*	*Gateway Processor*	*Controller Processor*	*Processing Engine*	*Spooler Processor*
102	Send query and report request to Gateway Processor	Receive query and report request			
104		Assign session handle to requests			
106		Assign Task ID to requests			
108		Return session identifier to client processor			
110		Send Task ID to	Receive Task ID		

		Controller Processor			
112		Send Task ID with report information to Spooler Processor	Share Task ID with other controller processors		Receive Task ID and report information
114		Send Query to Processing Engine(s)		Receive Query	
116				Process Query	
118		Receive complete message		Notify Gateway that task is done	
120		Notify Controller that task is done	Receive completion message		
122			Notify Spooler Processor that task is done		Receive completion message
124			Notify other controller processors that task is done		Request data from processor
126				Send data to spooler	Format report

128					Format Report
130					Output report

Table 2

Step Number	*Client Processor*	*Gateway Processor*	*Controller Processor*	*Processing Engine*
102	Send request to Gateway Processor		Receive Request	
104		Assign session handle to request		
106		Assign Task ID to request		
108		Send session handle to Client Processor		
110		Send Task ID to Controller Processor	Receive Task ID	
112		Send request to Processing Engine		Receive request
114				Lock target data
116				Process request
118				Unlock target data

120		Receive completion message		Notify Gateway Processor that task is done
122		Notify Controller Processor that task is done	Receive completion message	

CLAIMS

What is claimed is:

1. A processing system comprising:

a gateway processor;

a controller processor in communication with said gateway processor; and

a processing engine in communication with said gateway processor and said controller processor.

2. The system of claim 1 further comprising a client processor in communication with said gateway processor.

3. The system of claim 1 further comprising a spooler processor in communication with said gateway processor, said controller processor, and said processing engine.

4. The system of claim 3 wherein each of said spooler processor, said gateway processor, said controller processor, and said processing engine includes a statistics monitor.

5. The system of claim 3 wherein each of said spooler processor, said gateway processor, said controller processor, and said processing engine includes a watchdog monitor.

6. The system of claim 1 wherein said gateway processor, said controller processor, and said processing engine communicate by messages.

7. The system of claim 1 wherein the processing engine is a database engine.

8. The system of claim 1 wherein the processing engine is a computational engine.

9. The system of claim 1 wherein said system resides in at least one memory element of only one host machine.

10. The system of claim 1 wherein said system resides in at least one memory element of a plurality of host machines.

11. The system of claim 10 wherein said plurality of host machines are networked.

12. A memory element containing the system of claim 1 wherein said memory element is selected from the group of: hard disk, floppy disk, CD-ROM, optical disk, and magnetic tape.

13. A method for processing data comprising the steps of:

providing at least one gateway processor;

providing at least one controller processor;

receiving a transaction from a client processor by a gateway process;

distributing data from the at least one gateway processor to the at least one controller processor in response to the transaction.

14. The method of claim 13 wherein said method further comprises the steps of transmitting a transaction to the gateway process by a client processor.

15. The method of claim 13 further comprising the steps of:

providing at least one processing engine; and

distributing data from the at least one gateway processor to the at least one processing engine in response to the transaction.

16. The method of claim 13 further comprising the steps of:

providing at least one spooler processor; and

distributing data from the at least one gateway processor to the at least one spooler processor in response to the transaction.

17. A method for processing data comprising the steps of:

providing at least one gateway processor;

providing at least one controller processor;

providing at least one processing engine;

receiving a transaction from a client process with the at least one gateway processor;

transmitting a session handle associated with the transaction from the at least one gateway processor to the client process;

assigning a task identification number to the transaction;

transmitting information associated with the transaction from the at least one gateway processor to the at least one controller processor, wherein said information includes the task identification number;

transmitting information associated with the transaction from the at least one gateway processor to the at least one processing engine;

processing a plurality of data indicated by the transmitted information with the at least one processing engine;

transmitting a signal indicating completion of said processing step from the at least one processing engines to the at least one gateway controller; and

transmitting a signal indicating completion of said processing step from the at least one gateway processor to the at least one controller processor.

18. The method of claim 17 further comprising the steps of:

providing at least one spooler controller;

transmitting information associated with the transaction from the at least one gateway processor to the at least one spooler controller, wherein the information includes the task identification number;

requesting the processed information from the at least one processing engine by the at least one spooler controller; and

transmitting the requested information from the at least one spooler controller to an output device.

19. The method of claim 17 wherein said step of processing the information includes the step of locking out other accesses to a plurality of data indicated by the transmitted information.

20. A processing system comprising:

a processor having a process embodied therein wherein said process is selected from the group consisting of a gateway process, a controller process, a client process, a processing engine, and a spooler process, said process communicating with a gateway process, a controller process, a client process, a processing engine, and a spooler process.

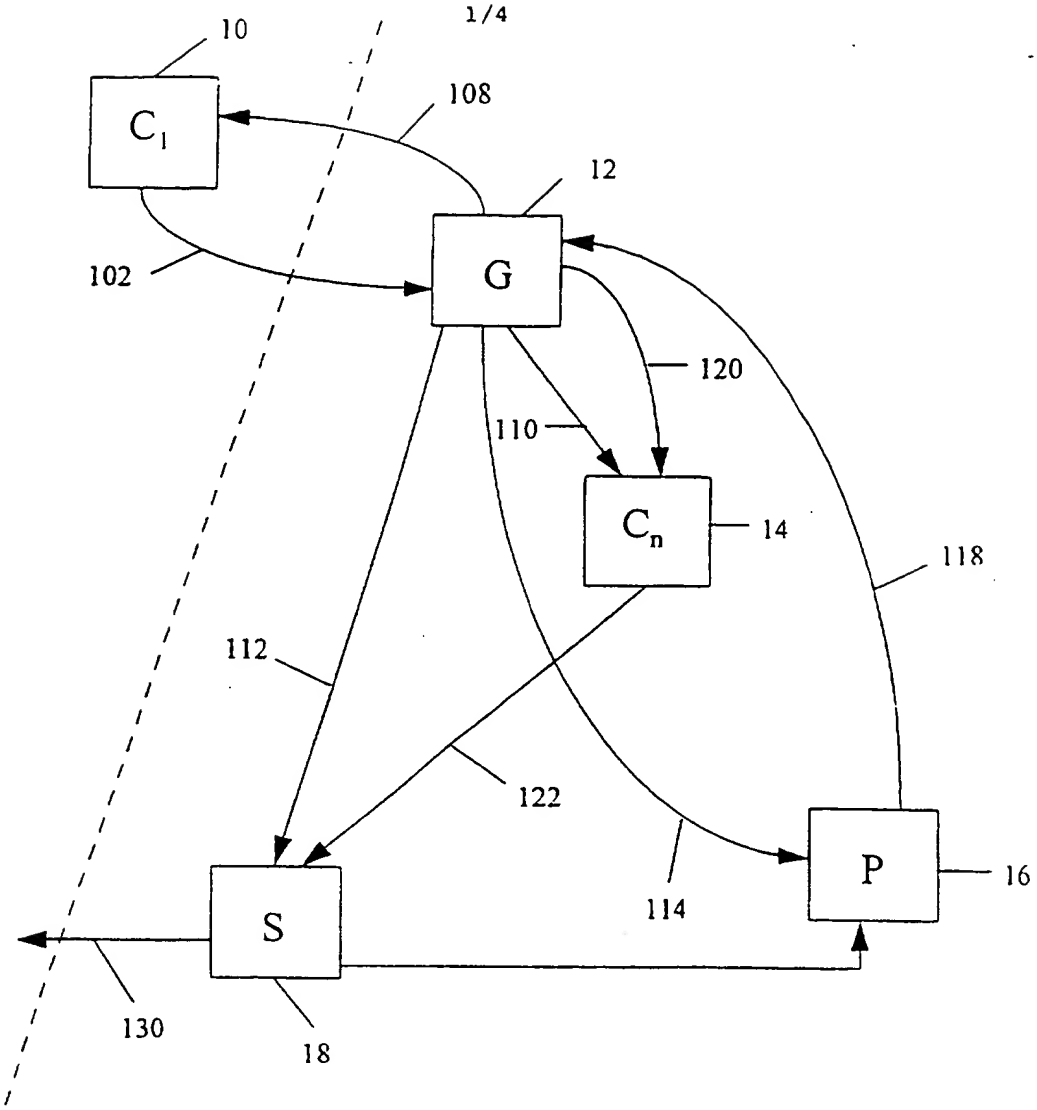

Fig. 1A

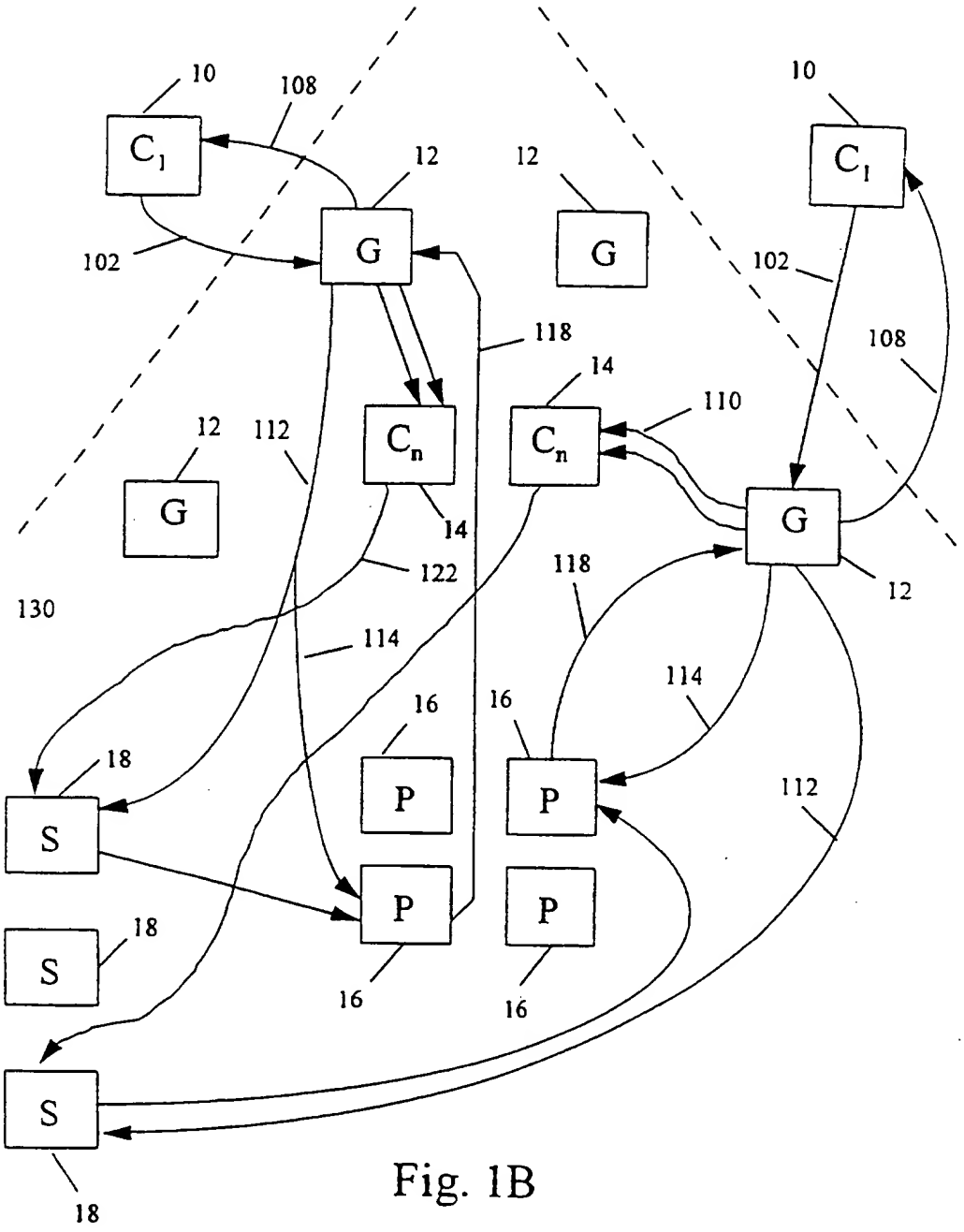

Fig. 1B

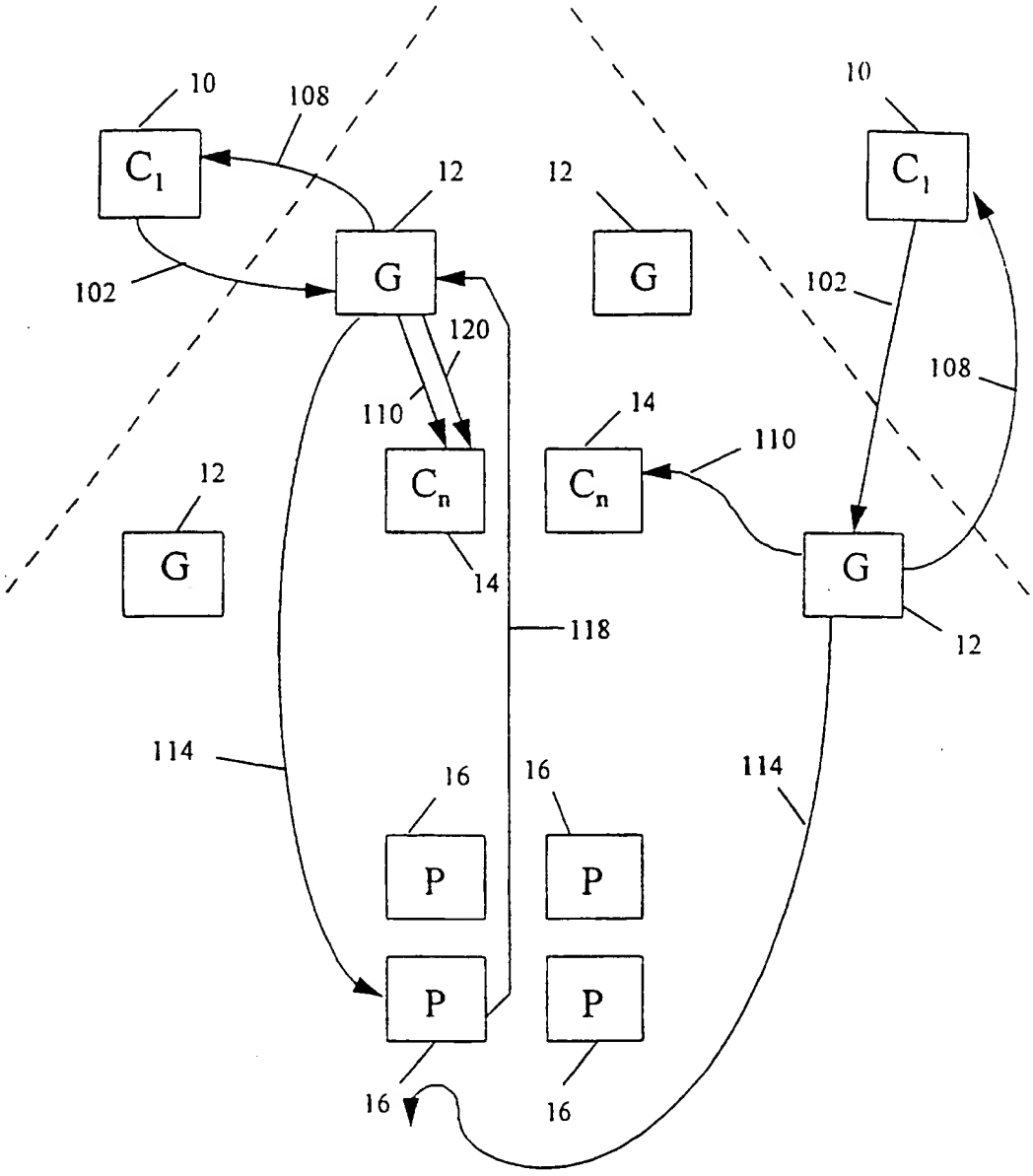

Fig. 2

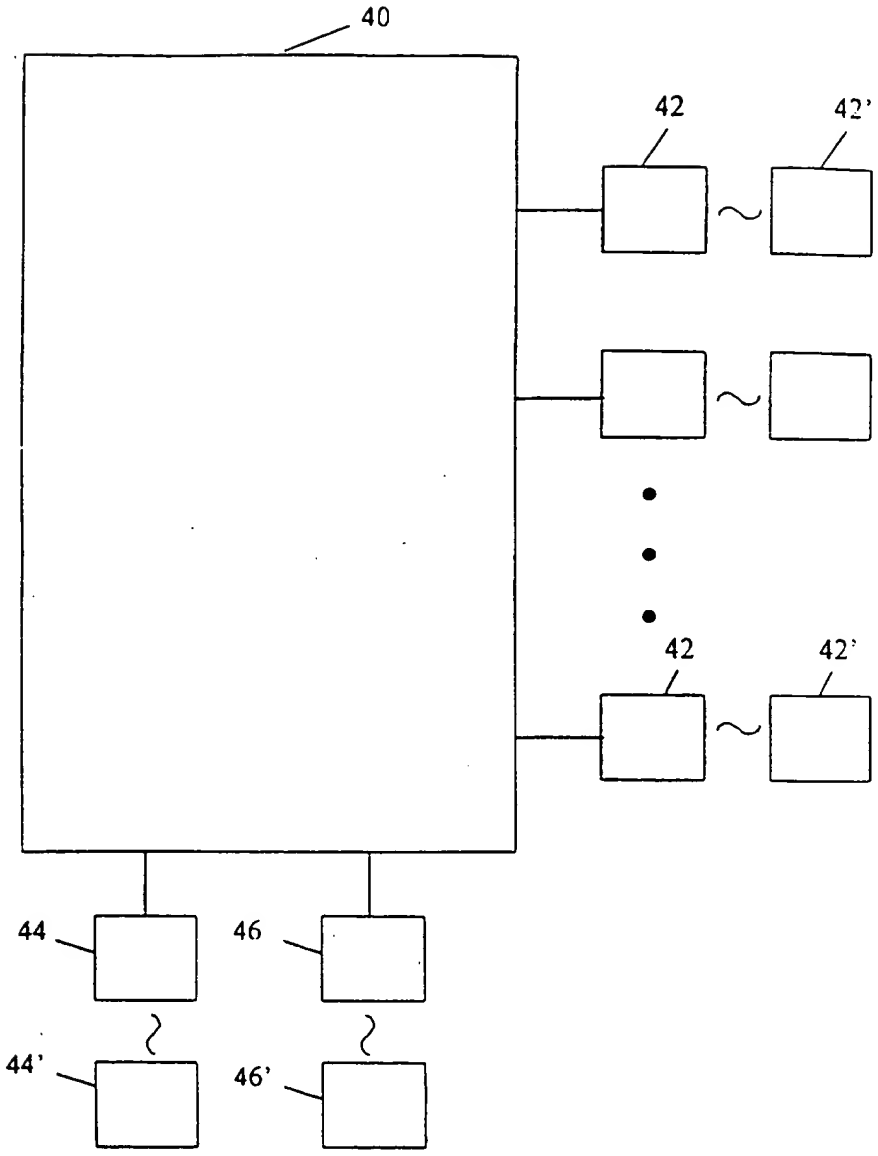

Fig. 3

INTERNATIONAL SEARCH REPORT

Inte nal Application No
PCT/US 97/02302

A. CLASSIFICATION OF SUBJECT MATTER
IPC 6 G06F9/46 H04L29/06

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 6 G06F H04L

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
Y	US 5 463 625 A (YASREBI MEHRAD) 31 October 1995 see column 3, line 5 - line 65 see column 5, line 3 - column 6, line 2 ---	1,2, 6-17,19
Y	EP 0 384 339 A (DIGITAL EQUIPMENT CORP) 29 August 1990 see the whole document ---	1,2, 6-17,19
X	MICROPROCESSING AND MICROPROGRAMMING, vol. 23, no. 1 - 05 + INDEX, March 1988, pages 279-281, XP000005724 MANASIEV L ET AL: "A SYSTEM SERVICE FOR DISTRIBUTED PROGRAMMING AND EXECUTION IN COMMUNICATION-ORIENTED DISTRIBUTED OPERATING MEDIA"	1-3, 6-13,20
A	see the whole document ---	14-19
	-/--	

[X] Further documents are listed in the continuation of box C.

[X] Patent family members are listed in annex.

* Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
5 June 1997	17. 06. 97

Name and mailing address of the ISA	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+ 31-70) 340-2040, Tx. 31 651 epo nl, Fax (+ 31-70) 340-3016	Michel, T

Form PCT/ISA/210 (second sheet) (July 1992)

INTERNATIONAL SEARCH REPORT

Ir onal Applicaon No
PCT/US 97/02302

C.(Conunuaon) DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citaon of document, with indicaon, where appropriate, of the relevant passages	Relevant to claim No.
A	IBM TECHNICAL DISCLOSURE BULLETIN, vol. 34, no. 5, 1 October 1991, pages 360-361, XP000189814 "LOCAL AREA NETWORK TO HOST PRINT SERVER" see the whole document ---	1-20
A	GB 2 210 482 A (LUSH ALAN;KENNY MICHAEL) 7 June 1989 see the whole document ---	4
A	PATENT ABSTRACTS OF JAPAN vol. 095, no. 003, 28 April 1995 & JP 06 334708 A (NEC CORP), 2 December 1994, see abstract -----	5

1

Form PCT/ISA/210 (conunuaon of second sheet) (July 1992)

INTERNATIONAL SEARCH REPORT

information on patent family members

Int nal Application No
PCT/US 97/02302

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 5463625 A	31-10-95	CA 2107299 A,C	30-03-95
		US 5596579 A	21-01-97
EP 0384339 A	29-08-90	AT 151183 T	15-04-97
		AU 611605 B	13-06-91
		AU 4996190 A	13-09-90
		AU 630291 B	22-10-92
		AU 7603391 A	15-08-91
		CA 2010762 A	24-08-90
		DE 69030340 D	07-05-97
		JP 3116262 A	17-05-91
		US 5341477 A	23-08-94
GB 2210482 A	07-06-89	NONE	

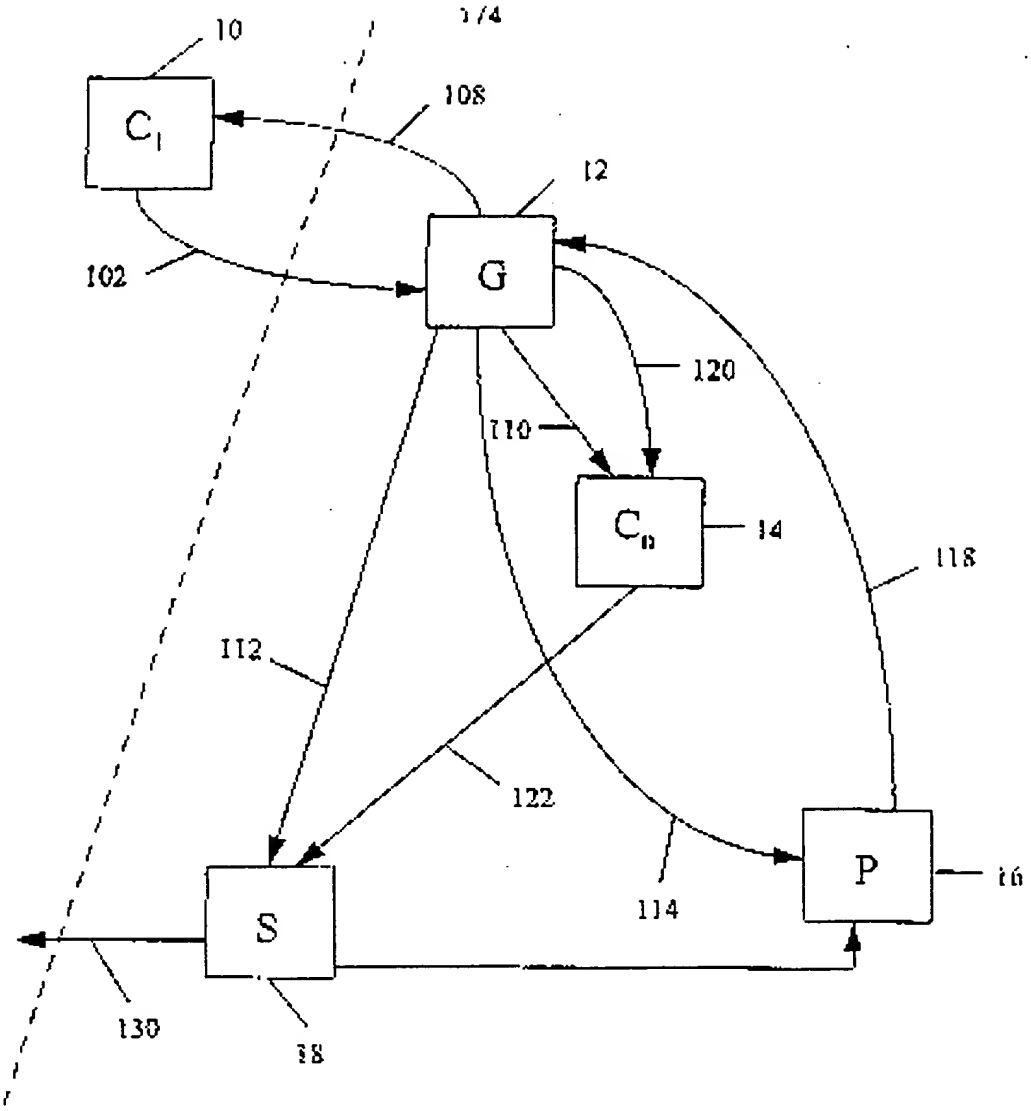

Fig. 1A

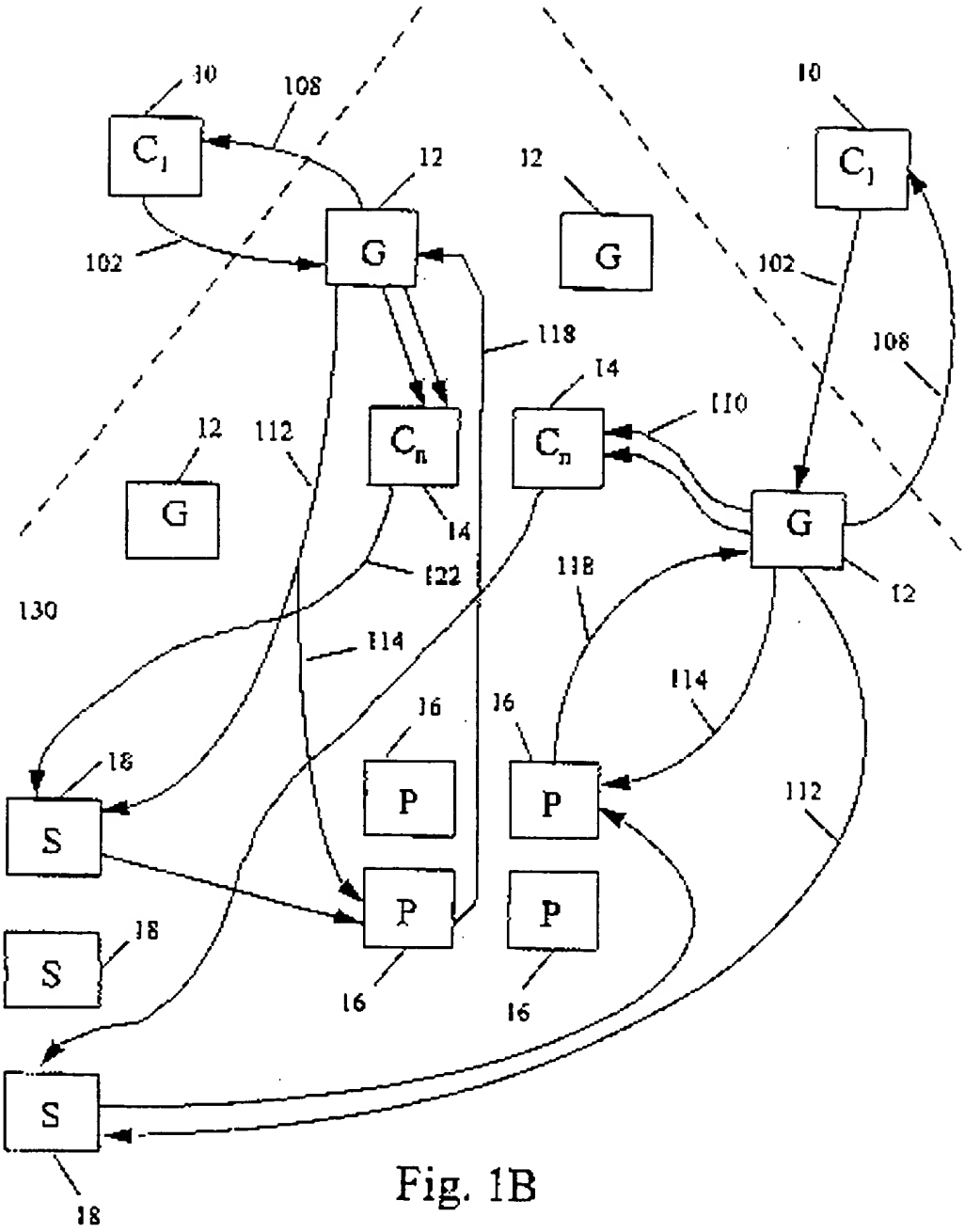

Fig. 1B

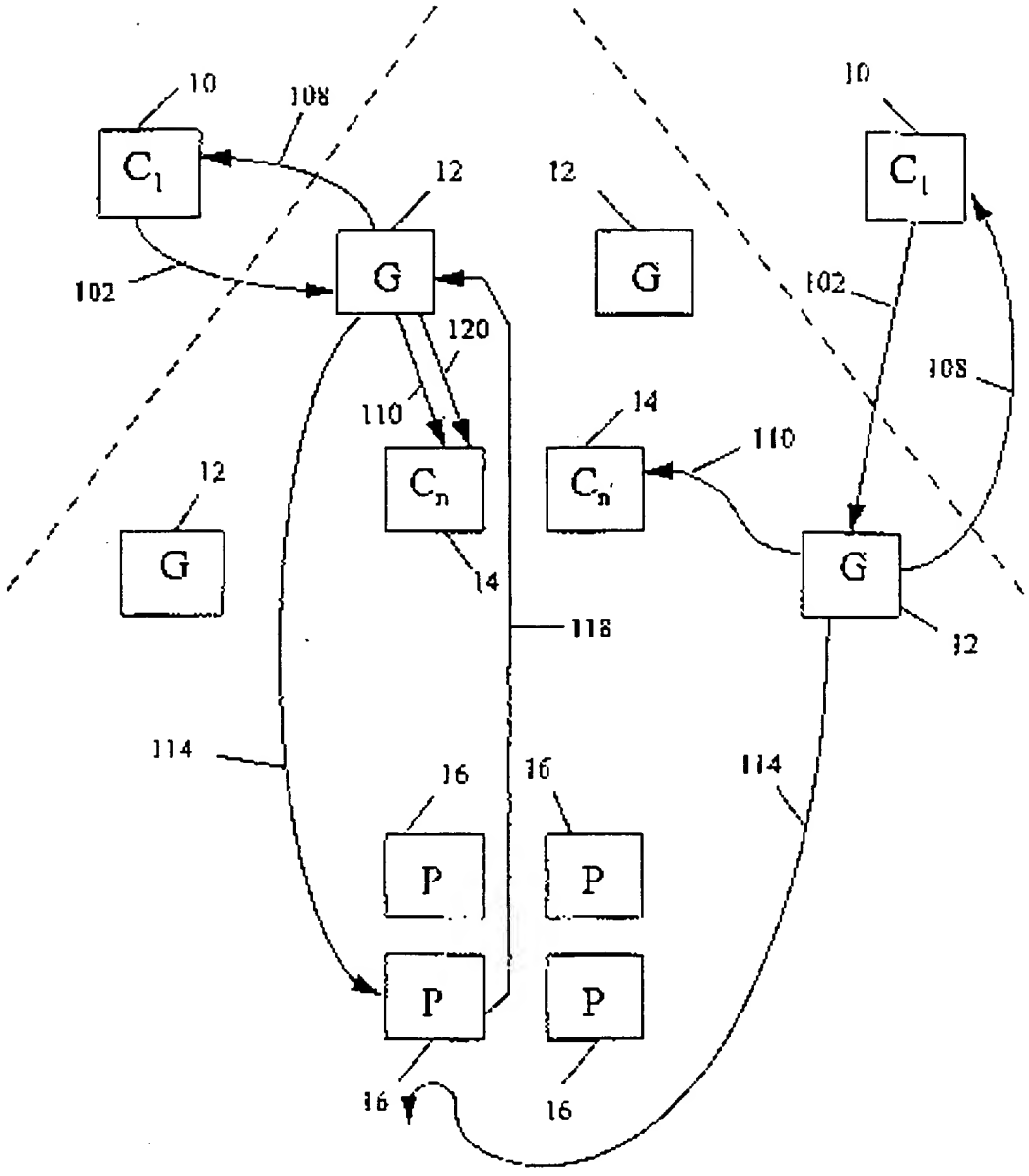

Fig. 2

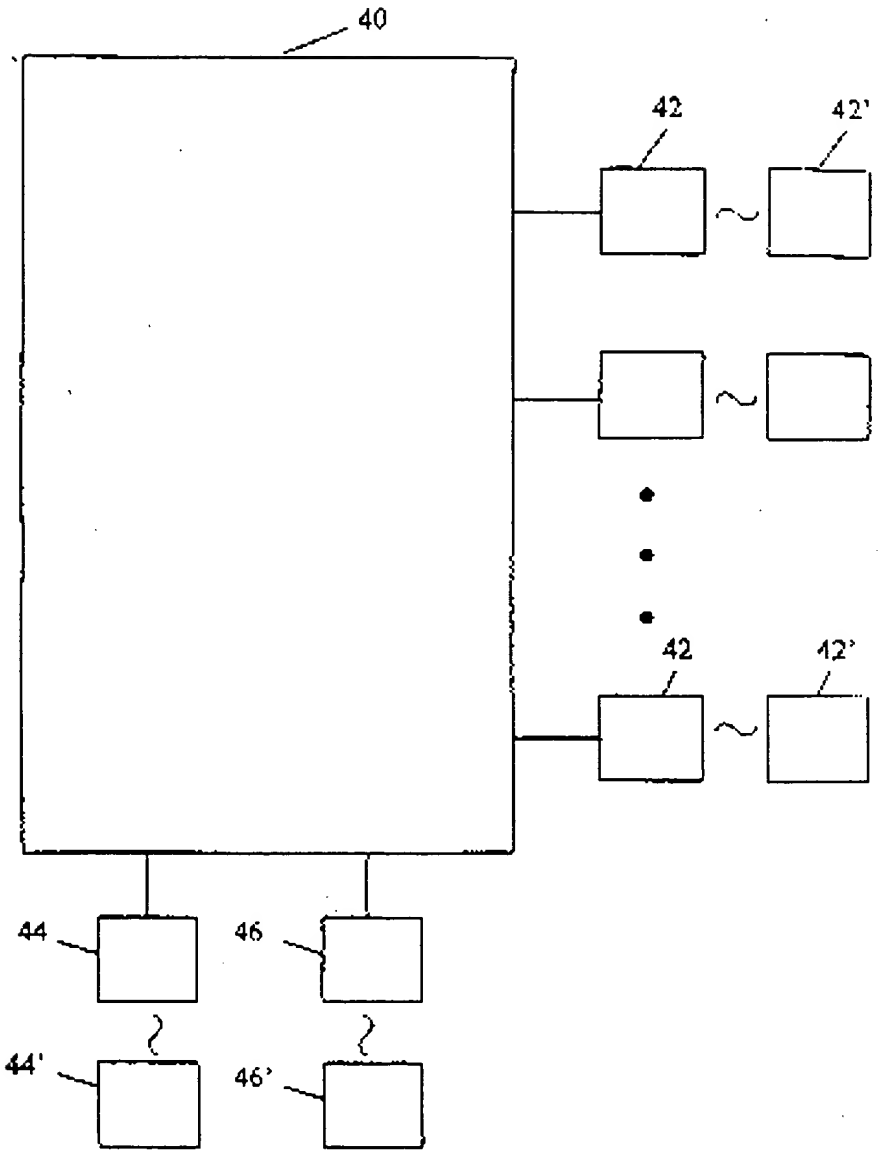

Fig. 3